目錄

声明

此书电子版免费供大家下载阅读，如果您已为此副本付费，请立即申请退款并联系作者举报此行为。请注意，虽然此书电子版免费供大家阅读，但这并不代表作者放弃了版权，您在未经授权的情况下依然不得以任何方式复制或抄袭本书内容。此书的电子版目前仅授权图灵社区和gitbook.com两个平台发布，如果您通过其他渠道获取到了此副本，则是侵权行为，请到上述两个平台下载合法授权的副本。获取合法授权副本的好处是可以及时得到此书的最新版本，早期版本中的错误会被及时纠正。感谢您对版权保护工作所做出的贡献。

作者邮箱:shijingjing02@163.com

作者Github: https://github.com/shijingjing1221/

Ansible介绍

什么是Ansible?

Ansilbe是一个部署一群远程主机的工具。远程的主机可以是本地或者远程的虚拟机，也可以是远程的物理机。

Ansible能做什么？

Ansilbe通过SSH协议进行管理节点和远程节点之间的通信。理论上说管理员通过ssh到一台远程主机上能做的操作Ansible都可以做。

包括：

- 拷贝文件
- 安装包
- 起服务
- ...

快速定位本书

Google "Ansible入门" 或 访问网站 Ansible入门 http://getansible.com/

本书资源

本文的所有ansible playbook例子都放在github上,欢迎补充和纠错:

https://github.com/ansible-book/ansible-first-book-examples

也可以联系作者进行纠正错误： shijingjing02@163.com

Ansible的架构

Ansilbe管理员节点和远程主机节点通过ssh协议进行通信。所以Ansible配置的时候只需要保证从Ansible管理节点通过SSH能够连接到被管理的远程的远程节点即可，当然需要建立的ssh，是基于key的，不能要求输入密码，下一章会讲到具体的配置方法。

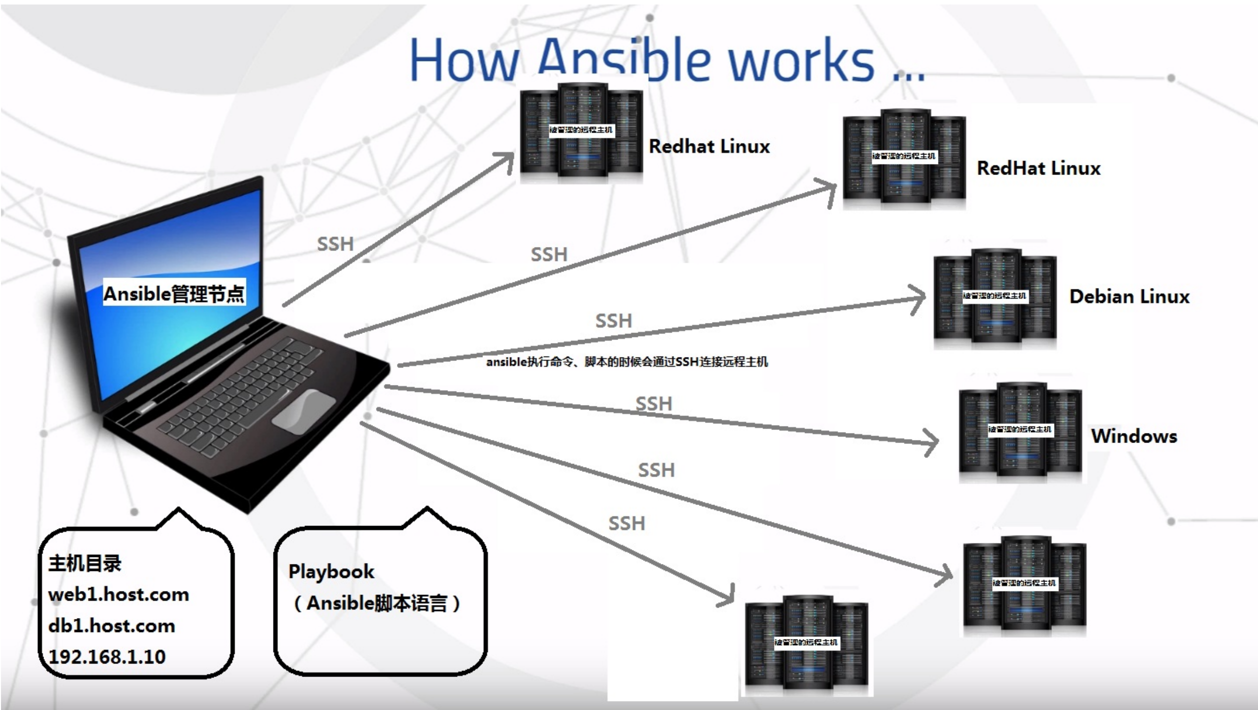
连接方式SSH

在管理员节点安装Ansible，编写脚本。在管理节点执行命令或者脚本时，通过SSH连接被管理的主机。 被管理的远程节点不需要进行特殊安装软件。



支持多种类型的主机

Ansible可以同时管理Redhat系的Linux，Debian系的Linux，以及Windows主机。管理节点只在执行脚本时与远程主机连接，没有特别的同步机制，所以发生断电等异常一般不会影响ansbile。


How Ansible works ...
Ansible管理节点
Redhat Linux
RedHat Linux
Debian Linux
Windows
SSH
SSH
SSH
SSH
SSH
SSH
ansible执行命令、脚本的时候会通过SSH连接远程主机
主机目录
web1.host.com
db1.host.com
192.168.1.10
Playbook
（Ansible脚本语言）

Ansible Tower的架构

为什么要有Ansbile Tower

Ansilbe Tower一款针对企业级的收费软件。

在上一节的Ansible架构中和下一章Ansbile的安装中会讲到，每一台被ansible远程管理的主机，都需要配置基于key的ssh连接。个人用户自己管理几台虚拟机和远程主机不会有什么问题，但是作为企业级也用户，则满足不了业务和安全上的需求。

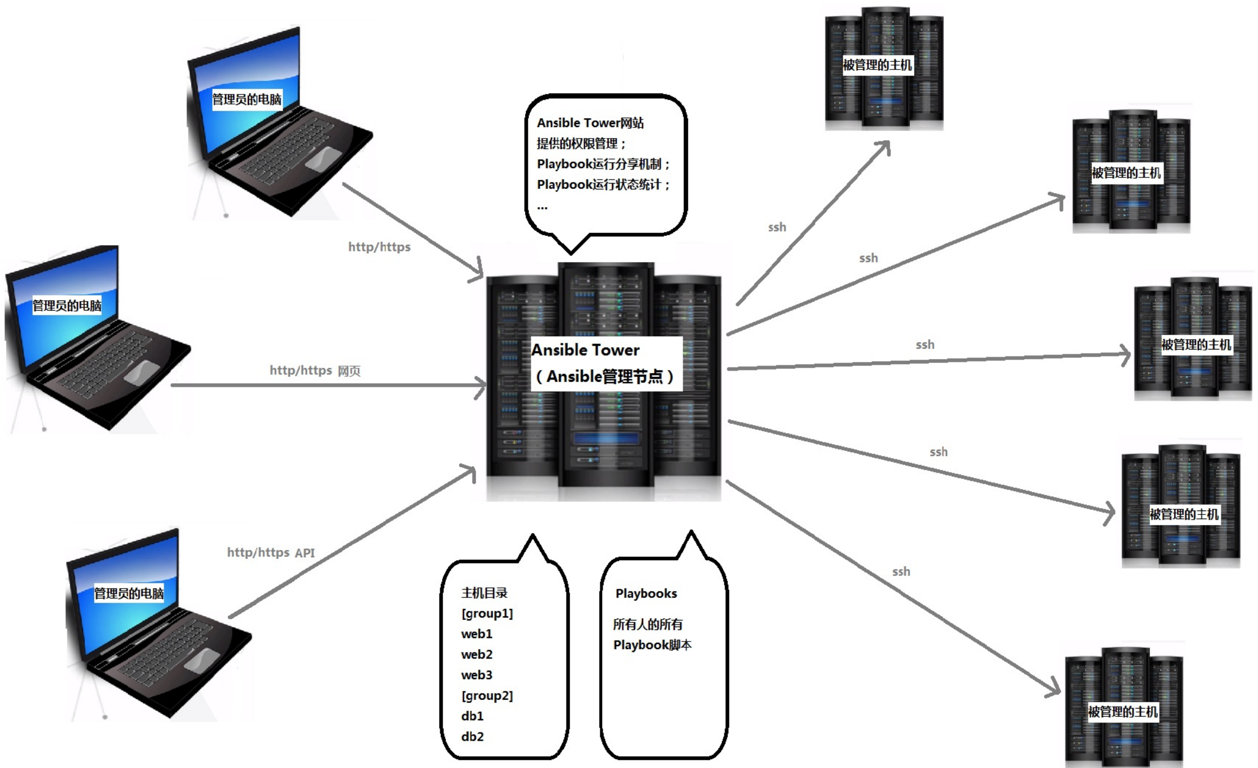
- 首先，每增加一台主机，都需要手工配置一下ssh连接，企业级的pc主机成百上千，每个管理员都需要在自己的电脑上配置所有的ssh连接，无疑工作量巨大。
- 还有，在安全方面如果管理员能够拿到ssh key，或者拷贝给别人，对于生产环境来说无疑是最大的安全隐患。

Ansible Tower能做什么

Ansile Tower则是针对解决企业级用户需求的，ansible tower是中心化ansible管理节点，它向管理员提供网站页面作为接口，来运行ansible脚本playbook。

- 管理员在ansible tower上使用和分享主机的ssh key，但是不能查看和拷贝key文件。
- ansible网站所有人可以共享playbook脚本，减少重复工作。
- 此外ansible还可以收集和展现所有主机的playbook的执行状况，便于统计和分析主机的状态。

说了这么多，看下面这张架构图就清晰了：


管理员的电脑
管理员的电脑
管理员的电脑
http/https
http/https 网页
http/https API
Ansible Tower网站
提供的权限管理；
Playbook运行分享机制；
Playbook运行状态统计；
...
Ansible Tower
（Ansible管理节点）
主机目录
[group1]
web1
web2
web3
[group2]
db1
db2
Playbooks
所有人的所有
Playbook脚本
ssh
ssh
ssh
ssh
ssh
被管理的主机
被管理的主机
被管理的主机
被管理的主机
被管理的主机

Ansible上手

怎么使用Ansible，本章通过简单的例子还说明Ansilbe上手的基本步骤。

1. 安装Ansible
2. Ansible管理哪些主机（主机目录管理）
3. 使用Ansilbe命令行管理主机（Ad-hoc command）
4. 使用Ansilbe脚本语言管理主机（脚本语言Playbook）
5. Ansible的“命令”Module

安装Ansile

这里以RedHat系Linux为例，其他系统请参考ansible的官网

管理员的电脑上：

- 安装Ansible软件

```
$ # Redhat/CentOS Linux上，Ansible目前放在的epel源中
$ # Fedora默认源中包含ansible，直接安装包既可
$ sudo yum install epel-release
$ sudo yum install ansible -y
```

- 配置Ansible管理节点和主机的连接

其实就是配置从管理节点到远程主机之间基于key（无密码的方式）的**SSH**连接：

```
$ # 生成ssh key
$ ssh-keygen
$ # 拷贝ssh key到远程主机，ssh的时候就不需要输入密码了
$ ssh-copy-id remoteuser@remoteserver
$ # ssh的时候不会提示是否保存key
$ ssh-keyscan remote_servers >> ~/.ssh/known_hosts
```

验证下有没有装好: 在管理节点执行下面的ssh命令，既不需要输入密码，也不会提醒你存储**key**，那就成功啦。

```
$ ssh remoteuser@remoteserver
```

被管理的远程主机：

不需要安装特殊的包，只需要python>2.4，RedHat Linux一般安装方式都自带。

Ansible的 Host Inventory

什么是 Host Inventory （主机目录、主机清单）？

Host Inventory 是配置文件，用来告诉Ansible需要管理哪些主机。并且把这些主机根据按需分类。

可以根据用途分类：数据库节点，服务节点等；根据地点分类：中部，西部机房。

Host Inventory 配置文件：

默认的文件是： **\/etc\/ansible\/hosts**

可以修改为其它的文件，下一章Ansible进阶中介绍。

例子

最简单的hosts文件：

```
192.168.1.50
aserver.example.org
bserver.example.org
```

带分类的hosts文件:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Ansible用命令管理主机

Ansible提供了一个命令行工具，在官方文档中起给命令行起了一个名字叫Ad-Hoc Commands。

ansible命令的格式是：

```
ansible <host-pattern> [options]
```

ansible命令功能有哪些

先不用深纠命令的语法，讲完module那节，就可以理解语法。先从感官上，通过下面的命令认识下ansible的命令行都可以做什么。

检查ansible安装环境

检查所有的server，是否以bruce用户创建了ansible主机可以访问的环境。

```
$ansible all -m ping -u bruce
```

执行命令

在所有的server上，以当前bash的同名用户，在远程主机执行“echo bash”

```
$ansible all -a "/bin/echo hello"
```

拷贝文件

拷贝文件/etc/host到远程机器（组）atlanta，位置为/tmp/hosts

```
$ ansible web -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

安装包

远程机器（组）webservers安装yum包

```
$ ansible web -m yum -a "name=acme state=present"
```

添加用户

```
$ ansible all -m user -a "name=foo password=<crypted password here>"
```

下载git包

```
$ ansible web -m git -a "repo=git://foo.example.org/repo.git dest=/srv/myapp version=HEAD"
```

起服务

```
$ ansible web -m service -a "name=httpd state=started"
```

并行执行

启动10个并行进行执行重起

```
$ansible lb -a "/sbin/reboot" -f 10
```

查看远程主机的全部系统信息！！！

```
$ ansible all -m setup
```

Ansible用脚本管理主机

只有脚本才可以重用，避免总敲重复的代码。Ansible脚本的名字叫Playbook，使用的是YAML的格式，文件以yml结尾。

注解：YAML和JSON类似，是一种表示数据的格式。

执行脚本playbook的方法

```
$ansible-palybook deploy.yml
```

playbook的例子

deploy.yml的功能为web主机部署apache, 其中包含以下部署步骤：

1. 安装apache包；
2. 拷贝配置文件httpd，并保证拷贝文件后，apache服务会被重启；
3. 拷贝默认的网页文件index.html；
4. 启动apache服务；

playbook deploy.yml包含下面几个关键字，每个关键字的含义：

- **hosts**：为主机的IP，或者主机组名，或者关键字all
- **remote_user**: 以哪个用户身份执行。
- **vars**： 变量
- **tasks**: playbook的核心，定义顺序执行的动作action。每个action调用一个ansbile module。
- > action 语法： `module： module_parameter=module_value`
- > 常用的module有yum、copy、template等，module在ansible的作用，相当于bash脚本中yum，copy这样的命令。下一节会介绍。
- **handers**： 是playbook的event，默认不会执行，在action里触发才会执行。多次触发只执行一次。

```
---
- hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest

  - name: Write the configuration file
    template: src=templates/httpd.conf.j2 dest=/etc/httpd/conf/h
ttpd.conf
    notify:
    - restart apache

  - name: Write the default index.html file
    template: src=templates/index.html.j2 dest=/var/www/html/ind
ex.html

  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

不懂yml，没关系，上面的deploy.yml格式转化为json格式为：

```
[
  {
    "hosts": "web",
    "vars": {
      "http_port": 80,
      "max_clients": 200
    },
    "remote_user": "root",
    "tasks": [
```

```
        {
          "name": "ensure apache is at the latest version",
          "yum": "pkg=httpd state=latest"
        },
        {
          "name": "Write the configuration file",
          "template": "src=templates/httpd.conf.j2 dest=/etc/httpd
/conf/httpd.conf",
          "notify": [
            "restart apache"
          ]
        },
        {
          "name": "Write the default index.html file",
          "template": "src=templates/index.html.j2 dest=/var/www/h
tml/index.html"
        },
        {
          "name": "ensure apache is running",
          "service": "name=httpd state=started"
        }
      ],
      "handlers": [
        {
          "name": "restart apache",
          "service": "name=httpd state=restarted"
        }
      ]
    }
]
```

提供json和yml互转的在线网站： http:\/\/www.json2yaml.com\/

Play VS Playbook

Playbook是指一个可被ansible执行的yml文件，一般的结构如下面的例子所示：

```
---
- hosts: web
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
```

其实在一个Playbook文件中还可以有针对两组server进行不同的操作，例如给web安装http服务器，和给lb安装mysql放在一个文件中：

```
---
#安装apache的play
- hosts: web
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest

# 安装mysql server的play
- hosts: lb
  remote_user: root
  tasks:
  - name: ensure mysqld is at the latest version
    yum: pkg=mariadb state=latest
```

像上面例子中针对每一组server的所有操作就组成一个play，一般一个playbook中只包含一个play，所以不用太在意区分playbook与play。如果在有些ansible文档中提到play的概念，你知道是怎么回事就可以了。

Ansible模块Module

什么是Ansible Module？

bash无论在命令行上执行，还是bash脚本中，都需要调用cd、ls、copy、yum等命令；module就是Ansible的“命令”，module是ansible命令行和脚本中都需要调用的。常用的Ansible module有yum、copy、template等。

在bash，调用命令时可以跟不同的参数，每个命令的参数都是该命令自定义的；同样，ansible中调用module也可以跟不同的参数，每个module的参数也都是由module自定义的。

每个module的用法可以查阅文档。http://docs.ansible.com/ansible/modules_by_category.html

Ansible在命令行里使用Module

在命令行中

> -m后面接调用module的名字
>
> -a后面接调用module的参数

```
$ #使用module copy拷贝管理员节点文件/etc/hosts到所有远程主机/tmp/hosts
$ ansible all -m copy -a "src=/etc/hosts dest=/tmp/hosts"
$ #使用module yum在远程主机web上安装httpd包
$ ansible web -m yum -a "name=httpd state=present"
```

Ansible在Playbook脚本使用Module

在playbook脚本中，tasks中的每一个action都是对module的一次调用。在每个action中：

> 冒号前面是module的名字
>
> 冒号后面是调用module的参数

```
---
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=templates/httpd.conf.j2 dest=/etc/httpd/conf/h
ttpd.conf
  - name: ensure apache is running
    service: name=httpd state=started
```

Module的特性

- 像Linux中的命令一样，Ansible的Module既上命令行调用，也可以用在Ansible的脚本Playbook中。
- 每个Module的参数和状态的判断，都取决于该module的具体实现，所以在使用他们之前都需要查阅该module对应的文档。
    - 可以通过文档查看具体的用法：http://docs.ansible.com/ansible/list_of_all_modules.html
    - 通过命令ansible-doc也可以查看module的用法
- Ansible提供一些常用功能的Module，同时Ansible也提供API，让用户可以自己写Module，使用的编程语言是Python。

介绍几个常用的module

学习Linux操作系统，如果不能一些基本的命令，那么真的没有办法用Linux。所以学习Ansible也非常有必要了解一些常用的module。

接下来介绍一些会在接下来的章节中用到的module，也是很常用的module。

调试和测试类的module

- ping - ping一下你的远程主机，如果可以通过ansible成功连接，那么返回pong
- debug - 用于调试的module，只是简单打印一些消息，有点像linux的echo命令。

文件类的module

- copy - 从本地拷贝文件到远程节点
- template - 从本地拷贝文件到远程节点，并进行变量的替换
- file - 设置文件的属性

linux上常用的操作

- user - 管理用户账户
- yum - red hat系linux上的包管理
- service - 管理服务
- firewalld - 管理防火墙中的服务和端口

执行Shell命令

- shell - 在节点上执行shell命令，支持$HOME和"<", ">", "|", ";" and "&"
- command - 在远程节点上面执行命令，不支持$HOME和"<", ">", "|", ";" and "&"

ping

这个就最常用的测试一个节点有没有配置好ssh连接的module。不过它可不是简单像linux命令中ping一下远程节点，而是首先检查下能不能SSH登陆，然后再检查下远程节点的python版本漫不满足要求，如果都满足则会返回成功pong。

ping不需要传入任何参数。因为ping是测试节点连接是不是通的，所以一般在命令行中使用的时候比在playbook的脚本中多，下面是ping在命令行中的用法：

```
ansible servers -m ping
```

debug

打印输出信息，和linux上的echo命令很像

通过参数msg定义打印的字符串

msg中可以嵌入变量,下面的例子中注入了系统变量，ansible在执行playbook之前会收集一些比较常用的系统变量，你在playbook中不需要定义直接就可以使用。

```
- debug:
    msg: "System {{ inventory_hostname }} has gateway {{ ansible
_default_ipv4.gateway }}"
```

执行结果

```
TASK [debug] *****************************************************
****************
ok: [localhost] => {
    "msg": "System localhost has gateway 192.168.50.1"
}
```

通过参数var定义需要打印的变量

变量可以是系统变量，也可以是动态的执行结果，通过关键字regester注入到变量中。

- 打印系统的变量

```
- name: Display all variables/facts known for a host
  debug:
    var: hostvars[inventory_hostname]["ansible_default_ipv4"]["gateway"]
```

执行结果：

```
TASK [Display part of variables/facts known for a host] ************************
ok: [localhost] => {
    "hostvars[inventory_hostname][\"ansible_default_ipv4\"][\"gateway\"]": "192.168.50.1"
}
```

- 打印动态注入的变量

```
- shell: /usr/bin/uptime
  register: result

- debug:
    var: result
```

执行结果

```
  TASK [command] ***********************************************
******************
  changed: [localhost]

  TASK [debug] *************************************************
******************
  ok: [localhost] => {
      "result": {
          "changed": true,

          "cmd": "/usr/bin/uptime",
          "delta": "0:00:00.003212",
          "end": "2017-01-01 21:30:02.817443",
          "rc": 0,
          "start": "2017-01-01 21:30:02.814231",
          "stderr": "",
          "stdout": " 21:30:02 up 12:38,  8 users,  load average
: 1.13, 1.31, 1.14",
          "stdout_lines": [
              " 21:30:02 up 12:38,  8 users,  load average: 1.13
, 1.31, 1.14"
          ],
          "warnings": []
      }
  }
```

copy

从当前的机器上copy静态文件到远程节点上，并且设置合理的文件权限。注意，copy module拷贝文件的时候，会先比较下文件的checksum，如果相同则不会拷贝，返回状态OK；如果不同才会拷贝，返回状态为changed。

设置文件权限

利用mode设置权限可以是用数字，当然也可以是符号的形式"u=rw,g=r,o=r"和"u+rw,g-wx,o-rwx"

```
- copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
    owner: foo
    group: foo
    mode: 0644
```

备份结点上原来的文件

backup参数为yes的时候，如果发生了拷贝操作，那么会先备份下目标节点山的原文件。当两个文件相同时，不会进行拷贝操作，当然也没有必要备份啦。

```
- copy:
    src: sudoers
    dest: /tmp
    backup: yes
```

拷贝后的验证操作

validate参数接需要验证的命令。一般需要验证拷贝后的文件，所以%s可以指代拷贝后的文件。当copy module中加入了validate参数，不仅需要成功拷贝文件，还需要validate命令返回成功的状态，整个module的执行状态才是成功。

`visudo -cf /etc/sudoers` 是验证sudoers文件有没有语法错误的命令。

```
- copy:
    src: /mine/sudoers
    dest: /etc/sudoers
    validate: 'visudo -cf %s'
```

template

如果你需要拷贝一个静态的文件，那么用copy module就够用了。但是如果你需要拷贝一个文件，并且根据需要部分内容，那么就需要用到template module啦。

比如安装apache后，你需要给节点拷贝一个测试页面index.html,index.html里面需要显示当前节点的主机名和IP,这时候就需要用到template。

index.html中，你需要指定你想替换的哪个部分，那么这个部分用变量来表示，template使用的是python的j2模版引擎，你不需要了解什么是j2，你只需要知道表量的表示法是 `{{}}` 就可以了。

template文件语法

index.html具体应该怎么写呢，既然是tamplate文件，那么我们就加一个后缀提高可读性，index.html.j2。下面文件中使用了两个变量ansible_hostname和ansible_default_ipv4.address。

```
<html>
<title>Demo</title>
<body>
<div class="block" style="height: 99%;">
    <div class="centered">
        <h1>#46 Demo</h1>
        <p>Served by {{ ansible_hostname }} ({{ ansible_default_
ipv4.address }}).</p>
    </div>
</div>
</body>
</html>
```

使用facts变量的template

在index.html.j2使用的两个变量ansible_hostname和ansible_default_ipv4.address都是facts变量，ansible会替我们搜索，直接可以在playbook中使用，当然也可以直接在template中使用。所以我们在写template语句中无需传入参数。

```
- name: Write the default index.html file
  template: src=templates/index.html.j2 dest=/var/www/html/index
.html
```

使用普通变量的template

拷贝httpd.conf.j2拷贝到远程节点，根据需求设置默认的http端口，这时我们就需要用到普通的变量。

在httpd.conf.j2模版文件中，所有变量的是用方法都是一样的，都是是用 `{{}}` :

```
ServerRoot "/etc/httpd"
...
Listen {{ http_port }}
...
```

普通变量不是在调用template的时候传进去，而是通过playbook中vars关键字定义。当然如果在playbook中可以直接使用的变量，都可以在template中，包括后面的章节会提到的定义在inventory中的变量。

```
- hosts: localhost
  vars:
    http_port: 8080
  remote_user: root
  tasks:

  - name: Write the configuration file
    template: src=templates/httpd.conf.j2 dest=/etc/httpd/conf/h
ttpd.conf
```

和copy module一样强大的功能

当然copy module不仅可以简单的拷贝文件到远程节点，还可以进行权限设置，文件备份，以及验证功能，那么这些功能，template同样具备。

```
- template:
    src: etc/ssh/sshd_config.j2
    dest: /etc/ssh/sshd_config.j2
    owner: root
    group: root
    mode: '0600'
    validate: /usr/sbin/sshd -t %s
    backup: yes
```

file

file module设置远程值机上的文件、软链接（symlinks）和文件夹的权限，也可以用来创建和删除他们。

改变文件的权限

当然mode参数可以直接赋值数字权限（必须以0开头），也可以赋值，还可以用来增加和删除权限。具体的写法见下面的代码：

```
- file:
    path: /etc/foo.conf
    owner: foo
    group: foo
    mode: 0644
    #mode: "u=rw,g=r,o=r"
    #mode: "u+rw,g-wx,o-rwx"
```

创建文件的软链接

注意这里面的src和sest参数的含义是和copy module不一样的，file module里面所操作的文件都是远程节点上的文件。

```
- file:
    src: /file/to/link/to
    dest: /path/to/symlink
    owner: foo
    group: foo
    state: link
```

创建一个新文件

像touch命令一样创建一个新文件

```
- file:
    path: /etc/foo.conf
    state: touch
    mode: "u=rw,g=r,o=r"
```

创建新的文件夹

```
# create a directory if it doesn't exist
- file:
    path: /etc/some_directory
    state: directory
    mode: 0755
```

user

user module可以增、删、改Linux远程节点的用户账户，并为其设置账户的属性。

增加账户

- 增加账户johnd，并且设置uid为1040，设置用户的primary group为admin

```
- user:
    name: johnd
    comment: "John Doe"
    uid: 1040
    group: admin
```

- 创建账户james，并为james用户额外添加两个group

```
- user:
    name: james
    shell: /bin/bash
    groups: admins,developers
    append: yes
```

删除账户

删除账户johnd

```
- user:
    name: johnd
    state: absent
    remove: yes
```

修改账户的属性

- 为账户jsmith撞见一个 2048-bit的SSH key，放在~jsmith/.ssh/id_rsa

```
- user:
    name: jsmith
    generate_ssh_key: yes
    ssh_key_bits: 2048
    ssh_key_file: .ssh/id_rsa
```

- 为用户添加过期时间：

```
- user:
    name: james18
    shell: /bin/zsh
    groups: developers
    expires: 1422403387
```

yum

yum module是用来管理red hat系的Linux上的安装包的，包括RHEL，CentOS，和fedora 21一下的版本。fedora从版本22开始就使用dnf，推荐使用dnf module来进行安装包的操作。

从yum源上安装和删除包

- 安装最新版本的包，如果已经安装了老版本，那么会更新到最新的版本：

```
- name: install the latest version of Apache
  yum:
    name: httpd
    state: latest
```

- 安装指定版本的包

```
- name: install one specific version of Apache
  yum:
    name: httpd-2.2.29-1.4.amzn1
    state: present
```

- 删除httpd包

```
- name: remove the Apache package
  yum:
    name: httpd
    state: absent
```

- 从指定的repo testing中安装包

```
- name: install the latest version of Apache from the testin
g repo
  yum:
    name: httpd
    enablerepo: testing
    state: present
```

从yum源上安装一组包

```
- name: install the 'Development tools' package group
  yum:
    name: "@Development tools"
    state: present

- name: install the 'Gnome desktop' environment group
  yum:
    name: "@^gnome-desktop-environment"
    state: present
```

从本地文件中安装包

```
- name: install nginx rpm from a local file
  yum:
    name: /usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch
.rpm
    state: present
```

从URL中安装包

```
- name: install the nginx rpm from a remote repo
  yum:
    name: http://nginx.org/packages/centos/6/noarch/RPMS/nginx-r
elease-centos-6-0.el6.ngx.noarch.rpm
    state: present
```

service

管理远程节点上的服务，什么是服务呢，比如httpd、sshd、nfs、crond等。

开、关、重起、重载服务

- 开httpd服务

  ```
  - service:
      name: httpd
      state: started
  ```

- 关服务

  ```
  - service:
      name: httpd
      state: stopped
  ```

- 重起服务

  ```
  - service:
      name: httpd
      state: restarted
  ```

- 重载服务

```
- service:
    name: httpd
    state: reloaded
```

设置开机启动的服务

```
- service:
    name: httpd
    enabled: yes
```

启动网络服务下的接口

```
- service:
    name: network
    state: restarted
    args: eth0
```

firewalld

firewalld module为某服务和端口添加firewalld规则。firewalld中有正在运行的规则，和永久的规则，firewalld module都支持。

firewalld要求远程节点上的firewalld版本在0.2.11以上。

为服务添加firewalld规则

```
- firewalld:
    service: https
    permanent: true
    state: enabled
```

```
- firewalld:
    zone: dmz
    service: http
    permanent: true
    state: enabled
```

为端口号添加firewalld规则

```
- firewalld:
    port: 8081/tcp
    permanent: true
    state: disabled
```

```
- firewalld:
    port: 161-162/udp
    permanent: true
    state: enabled
```

其它复杂的firewalld规则

```
- firewalld:
    rich_rule: 'rule service name="ftp" audit limit value="1/m"
accept'
    permanent: true
    state: enabled
```

```
- firewalld:
    source: 192.0.2.0/24
    zone: internal
    state: enabled
```

```
- firewalld:
    zone: trusted
    interface: eth2
    permanent: true
    state: enabled
```

```
- firewalld:
    masquerade: yes
    state: enabled
    permanent: true
    zone: dmz
```

shell

通过/bin/sh在远程节点上执行命令。如果一个操作你可以通过Module yum，copy操作实现，那么你不要使用shell或者command这样通用的命令module。因为通用的命令module不会根据具体操作的特点进行status的判断，所以当没有必要再重新执行的时候，它还是要重新执行一遍。

支持$home，支持$HOME和"<", ">", "|", ";" and "&"。

- 支持$home

  ```
  - name: test $home
    shell: echo "Test1" > ~/tmp/test1
  ```

- 支持&&

  ```
  - shell: service jboss start && chkconfig jboss on
  ```

- 支持>>

  ```
  - shell: echo foo >> /tmp/testfoo
  ```

调用脚本

- 调用脚本

```
- shell: somescript.sh >> somelog.txt
```

- 执行命令前，改变工作目录

```
- shell: somescript.sh >> somelog.txt
  args:
    chdir: somedir/
```

- 在执行命令钱改变工作目录，并且在文件somelog.txt不存在时执行命令。

```
- shell: somescript.sh >> somelog.txt
  args:
    chdir: somedir/
    creates: somelog.txt
```

- 指定用bash运行命令

```
- shell: cat < /tmp/\*txt
  args:
    executable: /bin/bash
```

Command

在远程节点上执行命令。和Shell Module类似，不过不支持$HOME and operations like "<", ">", "|", ";" and "&"。

和Shell一样

- 像Shell一样调用单条命令

```
- command: /sbin/shutdown -t now
```

- 和Shell一样，可以在执行命令钱改变目录，并检查文件database不存在时再执行

```
- command: /usr/bin/make_database.sh arg1 arg2
  args:
    chdir: somedir/
    creates: /path/to/database
```

和Shell不一样

- 与Shell不同，多了一个传参方式：

```
- command: /usr/bin/make_database.sh arg1 arg2 creates=/path
/to/database
```

- 不支持&&和>>

  下面的写法，没有办法创建~/tmp/test3和~/tmp/test4的

```
- name: test $home
  command: echo "test3" > ~/tmp/test3 && echo "test4" > ~/tm
p/test4
```

Ansible进阶

深入介绍一下几个主题

- Ansible的配置
- Ansible的主机目录管理(Host Inventory)
- Ansible Playbook的进阶语法
- 配置Extra Modules

ansible的配置

可以配置什麼?

从基本的，主机目录文件"inventory"，extra module放置路径"library" ，远程主机的临时文件位置" remote_tmp" ，管理节点上临时文件的位置"local_tmp"

```
inventory       = /etc/ansible/hosts
library         = /usr/share/my_modules/
remote_tmp      = $HOME/.ansible/tmp
local_tmp       = $HOME/.ansible/tmp
```

到高级的，连接端口号"accelerate_port"，超时时间等。

```
accelerate_port = 5099
accelerate_timeout = 30
accelerate_connect_timeout = 5.0
```

看一个完整的anbile配置文件例子，就能基本了解到ansible都能配置什么了:

https:\/\/raw.githubusercontent.com\/ansible\/ansible\/devel\/examples\/ansible.cfg

对ansible配置文件里面的关键字不能完整理解，还可以参考关键词解释列表:

http:\/\/docs.ansible.com\/ansible\/intro_configuration.html#explanation-of-values-by-section

anbile配置文件的优先级

ansible的默认配置文件是\/etc\/ansible\/ansible.cfg。其实ansible会按照下面的顺序查找配置文件，并使用第一个发现的配置文件。

```
* ANSIBLE_CONFIG (an environment variable)
* ansible.cfg (in the current directory)
* .ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg
```

Ansible1.5以前的版本顺序为：

```
* ansible.cfg (in the current directory)
* ANSIBLE_CONFIG (an environment variable)
* .ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg
```

主机目录(Host Inventory)

什么叫主机目录管理,告诉ansible需要管理哪些server，和server的分类和分组信息。可以根据你自己的需要根据地域分类，也可以按照功能的不同分类。

主机目录的配置文件

默认文件

/etc/ansible/hosts

修改主机目录的配置文件

/etc/ansible/ansible.cfg

```
...
inventory      = /etc/ansible/hosts
...
```

命令行中传递主机目录配置文件

```
$ ansible-playbook -i hosts site.yml
```

或者参数--inventory-file

```
$ ansible-playbook --inventory-file hosts site.yml
```

远程主机的分组

简单的分组[]内是组名

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com

[webservers]
www[01:50].example.com

[databases]
db-[a:f].example.com
```

分组usa的子组还可以是其它的组,例如[usa:children]中还可以子组southeast, [southeast:children]中还可以包含atlanta和releigh

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh


[usa:children]
southeast
northeast
southwest
northwest
```

指定连接的参数

参数

指定Server的连接参数，其中包括连接方法，用户等。

```
[targets]

localhost              ansible_connection=local
other1.example.com     ansible_connection=ssh        ansible_use
r=mpdehaan
other2.example.com     ansible_connection=ssh        ansible_use
r=mdehaan


[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

所有可以指定的参数在文档中
http://docs.ansible.com/ansible/intro_inventory.html#list-of-behavioral-inventory-parameters

变量

为一个组指定变量

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

按目录结构存储变量

假设inventory文件为/etc/ansible/hosts，那么相关的hosts和group变量可以放在下面的目录结构下

```
/etc/ansible/group_vars/raleigh # can optionally end in '.yml',
'.yaml', or '.json'
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/foosball
```

/etc/ansible/group_vars/raleigh 文件内容可以为

```
---
ntp_server: acme.example.org
database_server: storage.example.org
```

如果对应的名字为目录名，ansible会读取这个目录下面所有文件的内容

```
/etc/ansible/group_vars/raleigh/db_settings
/etc/ansible/group_vars/raleigh/cluster_settings
```

group_vars/ 和 host_vars/ 目录可放在 inventory 目录下,或是 playbook 目录下. 如果两个目录下都存在,那么 **playbook** 目录下的配置会覆盖 inventory 目录的配置.

Playbook

YML

ansible的脚本语言，yaml格式。请参考YAML语法结构章节

别人的Playbook

能够学会快速用别人的成果，高效地分享自己的成果，才是好码农，才是能早下班的好码农。在你动手从头开始写一个Playbook之前，不如先参考一下别人的成果吧。

官方例子

Ansible官方提供了一些比较常用的、经过测试的Playbook例子：

https:\/\/github.com\/ansible\/ansible-examples

Playbook分享平台

此外，Ansible还提供了Playbook的分享平台，上面的例子是Ansible用户自己上传的。你如果在没有思路的情况下参考下吧，不过一定要再重新严谨的测试下。

https:\/\/galaxy.ansible.com\/

Playbook基本语法

本节列举了写第一个Playbook，你必须了解基本语法。

随着你面临的机器越多，配置的需求越复杂，你可能需要了解后面介绍的一些稍微复杂逻辑的语句。

执行Playbook语法

```
$ ansible-playbook deploy.yml
```

查看输出的细节

```
ansible-playbook playbook.yml  --verbose
```

查看该脚本影响哪些hosts

```
ansible-playbook playbook.yml --list-hosts
```

并行执行脚本

```
ansible-playbook playbook.yml -f 10
```

完整的playbook脚本示例

最基本的playbook脚本分为三个部分:

1. 在什么机器上以什么身份执行
    - hosts
    - users
    - ...
2. 执行的任务是都有什么

  - tasks

3. 善后的任务都有什么

  - handlers

deploy.yml文件

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

主机和用户

key	含义
hosts	为主机的IP，或者主机组名，或者关键字all
user	在远程以哪个用户身份执行。
become	切换成其它用户身份执行，值为yes或者no
become_method	与became一起用，指可以为‘sudo’/’su’/’pbrun’/’pfexec’/’doas’
become_user	与bacome_user一起用，可以是root或者其它用户名

脚本里用became的时候，执行的playbook的时候可以加参数--ask-become-pass

```
ansible-playbook deploy.yml --ask-become-pass
```

Tasks任务列表

- tasks是从上到下顺序执行，如果中间发生错误，那么整个playbook会中止。你可以改修文件后，再重新执行。
- 每一个task的对module的一次调用。使用不同的参数和变量而已。
- 每一个task最好有name属性，这个是供人读的，没有实际的操作。然后会在命令行里面输出，提示用户执行情况。

语法

task的基本写法:

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=running
```

其中name是可选的，也可以简写成下面的例子。

```
tasks:
  - service: name=httpd state=running
```

写name的task在playbook执行时，会显示对应的名字，信息更友好、丰富。写name是个好习惯！

```
TASK: [make sure apache is running] ****************************
*********************************
changed: [yourhost]
```

没有写name的task在playbook执行时，直接显示对应的task语法。在调用同样的module多次后，不同意分辨执行到哪步了。

```
TASK: [service name=httpd state=running] **********************
***************
changed: [yourhost]
```

参数的不同写法

最上的代码展示了最基本的传入module的参数的方法 `key=value`

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=running
```

当需要传入参数列表太长时，可以分隔到多行：

```
 tasks:
  - name: Copy ansible inventory file to client
    copy: src=/etc/ansible/hosts dest=/etc/ansible/hosts
            owner=root group=root mode=0644
```

或者用yml的字典传入参数

```
 tasks:
  - name: Copy ansible inventory file to client
    copy:
      src: /etc/ansible/hosts
      dest: /etc/ansible/hosts
      owner: root
      group: root
      mode: 0644
```

TASK的执行状态

task中每个action会调用一个module，在module中会去检查当前系统状态是否需要重新执行。

- 如果本次执行了，那么action会得到返回值changed;
- 如果不需要执行，那么action得到返回值ok

module的执行状态的具体判断规则由各个module自己决定和实现的。例如，"copy" module的判断方法是比较文件的checksum，代码如下：

https:\/\/github.com\/ansible\/ansible-modules-core\/blob\/devel\/files\/copy.py

状态示例

以一个copy文件的task为例子:

```
tasks:
- name: Copy the /etc/hosts
  copy: src=/etc/hosts dest=/etc/hosts
```

第一次执行,它的结果是这个样子的:

TASK的状态是changed

```
[jshi@jshi demoansible]$ ansible-playbook copy_hosts.yml

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [vm-rhel7-1]
ok: [vm-rhel7-3]
ok: [vm-rhel7-2]

TASK [Copy the /etc/hosts] *****************************************************
changed: [vm-rhel7-3]
changed: [vm-rhel7-1]
changed: [vm-rhel7-2]

PLAY RECAP *********************************************************************
vm-rhel7-1                 : ok=2    changed=1    unreachable=0    failed=0
vm-rhel7-2                 : ok=2    changed=1    unreachable=0    failed=0
vm-rhel7-3                 : ok=2    changed=1    unreachable=0    failed=0
```

第二次执行是下面这个样子的:

TASK的状态是ok,由于第一次执行copy_hosts.yml的时候,已经拷贝过文件,那么ansible目标文件的状态避免重复执行.

```
[jshi@jshi demoansible]$ ansible-playbook copy_hosts.yml

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [vm-rhel7-3]
ok: [vm-rhel7-1]
ok: [vm-rhel7-2]

TASK [Copy the /etc/hosts] *****************************************************
ok: [vm-rhel7-1]
ok: [vm-rhel7-3]
ok: [vm-rhel7-2]

PLAY RECAP *********************************************************************
vm-rhel7-1                 : ok=2    changed=0    unreachable=0    failed=0
vm-rhel7-2                 : ok=2    changed=0    unreachable=0    failed=0
vm-rhel7-3                 : ok=2    changed=0    unreachable=0    failed=0
```

下面我更改vm-rhel7-1的\/etc\/hosts, 再次执行看看:

```
[jshi@jshi demoansible]$ ansible-playbook copy_hosts.yml

PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [vm-rhel7-1]
ok: [vm-rhel7-3]
ok: [vm-rhel7-2]

TASK [Copy the /etc/hosts] *****************************************************
ok: [vm-rhel7-2]
ok: [vm-rhel7-3]
changed: [vm-rhel7-1]

PLAY RECAP *********************************************************************
vm-rhel7-1                 : ok=2    changed=1    unreachable=0    failed=0
vm-rhel7-2                 : ok=2    changed=0    unreachable=0    failed=0
vm-rhel7-3                 : ok=2    changed=0    unreachable=0    failed=0
```

响应事件Handler

什么是handler?

每个主流的编程语言都会有event机制，那么handler就是playbook的event。

Handlers里面的每一个handler，也是对module的一次调用。而handlers与tasks不同，tasks会默认的按定义顺序执行每一个task，handlers则不会，它需要在tasks中被调用，才有可能被执行。

Tasks中的任务都是有状态的，changed或者ok。 在Ansible中，只在task的执行状态为changed的时候，才会执行该task调用的handler，这也是handler与普通的event机制不同的地方。

应用场景

什么情况下使用handlers呢?

如果你在tasks中修改了apache的配置文件。需要重起apache。此外还安装了apache的插件。那么还需要重起apache。像这样的应该场景中，重起apache就可以设计成一个handler.

一个handler最多只执行一次

在所有的任务里表执行之后执行，如果有多个task notify同一个handler,那么只执行一次。

在下面的例子里apache只执行一次

https://github.com/ansible-book/ansible-first-book-examples/blob/master/handlers_state.yml

```
---
- hosts: lb
  remote_user: root
  vars:
      random_number1: "{{ 10000 | random }}"
      random_number2: "{{ 10000000000 | random }}"
  tasks:
  - name: Copy the /etc/hosts to /tmp/hosts.{{ random_number1 }}
    copy: src=/etc/hosts dest=/tmp/hosts.{{ random_number1 }}
    notify:
      - call in every action
  - name: Copy the /etc/hosts to /tmp/hosts.{{ random_number2 }}
    copy: src=/etc/hosts dest=/tmp/hosts.{{ random_number2 }}
    notify:
      - call in every action

  handlers:
  - name: call in every action
    debug: msg="call in every action, but execute only one time"
```

action是Changed ,才会执行handler

只有当TASKS种的action的执行状态是changed时，才会触发notify handler的执行。

下面的脚本执行两次,执行结果是不同的:

- 第一次执行是，tasks的状态都是changed，会触发两个handler
- 第二次执行是,
    - 第一个task的状态是OK，那么不会触发handlers"call by /tmp/hosts",
    - 第二个task的状态是changed，触发了handler"call by /tmp/hosts.random_number"

测试代码见：

https://github.com/shijingjing1221/ansible-first-book-examples/blob/master/handlers_execution_time.yml

```
---
- hosts: lb
  remote_user: root
  vars:
      random_number: "{{ 10000 | random }}"
  tasks:
  - name: Copy the /etc/hosts to /tmp/hosts
    copy: src=/etc/hosts dest=/tmp/hosts
    notify:
      - call by /tmp/hosts
  - name: Copy the /etc/hosts to /tmp/hosts.{{ random_number }}
    copy: src=/etc/hosts dest=/tmp/hosts.{{ random_number }}
    notify:
      - call by /tmp/hosts.random_number

  handlers:
  - name: call by /tmp/hosts
    debug: msg="call first time"
  - name: call by /tmp/hosts.random_number
    debug: msg="call by /tmp/hosts.random_number"
```

按Handler的定义顺序执行

handlers是按照在handlers中定义个顺序执行的，而不是安装notify的顺序执行的。

下面的例子定义的顺序是1>2>3，notify的顺序是3>2>1，实际执行顺序：1>2>3.

```
---
- hosts: lb
  remote_user: root
  gather_facts: no
  vars:
      random_number1: "{{ 10000 | random }}"
      random_number2: "{{ 10000000000 | random }}"
  tasks:
  - name: Copy the /etc/hosts to /tmp/hosts.{{ random_number1 }}
    copy: src=/etc/hosts dest=/tmp/hosts.{{ random_number1 }}
    notify:
      - define the 3nd handler
  - name: Copy the /etc/hosts to /tmp/hosts.{{ random_number2 }}
    copy: src=/etc/hosts dest=/tmp/hosts.{{ random_number2 }}
    notify:
      - define the 2nd handler
      - define the 1nd handler

  handlers:
  - name: define the 1nd handler
    debug: msg="define the 1nd handler"
  - name: define the 2nd handler
    debug: msg="define the 2nd handler"
  - name: define the 3nd handler
    debug: msg="define the 3nd handler"
```

变量

定义

使用YAML格式定义

```
foo:
  field1: one
  field2: two
```

使用变量

使用Python的template语言Jinja2的语法引用：利用中括号和点号来访问子属性

```
foo['field1']
foo.field1
```

Playbook中使用的变量

在Playbook中使用，需要用{{ }}引用以来即可：

```
- hosts: webservers
  vars:
      apache_config: labs.conf
  tasks:
      - name: deploy haproxy config
        template: src={{ apache_config }} dest=/etc/httpd/conf.d
/{{ apache_config }}
```

在Playbook中使用变量文件定义变量

```
- hosts: webservers
  vars_files:
      - vars/server_vars.yml
  tasks:
      - name: deploy haproxy config
        template: src={{ apache_config }} dest=/etc/httpd/conf.d
/{{ apache_config }}
```

变量文件vars/server_vars.yml的内容为：

```
apache_config: labs.conf
```

YAML的陷阱

YAML的陷阱是YAML和Ansible Playbook的变量语法不能在一起好好工作了。这里特指冒号后面的值不能以"{ "开头。

下面的代码会报错:

```
- hosts: app_servers
  vars:
      app_path: {{ base_path }}/22
```

解决办法：要在"{ "开始的值加上引号:

```
- hosts: app_servers
  vars:
       app_path: "{{ base_path }}/22"
```

主机的系统变量(facts)

ansible会通过module setup来收集主机的系统信息，这些收集到的系统信息叫做facts，这些facts信息可以直接以变量的形式使用。

哪些facts变量可以引用呢？在命令行上通过调用setup module命令可以查看

```
$ ansible all -m setup -u root
```

怎样在playbook中使用facts变量呢，答案是直接使用：

```
---
- hosts: all
  user: root
  tasks:
  - name: echo system
    shell: echo {{ ansible_os_family }}
  - name install ntp on Debian linux
    apt: name=git state=installed
    when: ansible_os_family == "Debian"
  - name install ntp on redhat linux
    yum: name=git state=present
    when: ansible_os_family == "RedHat"
```

使用复杂facts变量

一般在系统中收集到如下的信息，复杂的、多层级的facts变量如何使用呢？

```
...
        "ansible_ens3": {
            "active": true,
            "device": "ens3",
            "ipv4": {
                "address": "10.66.192.234",
                "netmask": "255.255.254.0",
                "network": "10.66.192.0"
            },
            "ipv6": [
                {
                    "address": "2620:52:0:42c0:5054:ff:fef2:e2a3
",
                    "prefix": "64",
                    "scope": "global"
                },
                {
                    "address": "fe80::5054:ff:fef2:e2a3",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "52:54:00:f2:e2:a3",
            "module": "8139cp",
            "mtu": 1500,
            "promisc": false,
            "type": "ether"
        },
...
```

那么可以通过下面的两种方式访问复杂的变量中的子属性:

中括号：

```
{{ ansible_ens3["ipv4"]["address"] }}
```

点号：

```
{{ ansible_ens3.ipv4.address }}
```

关闭facts

在Playbook中,如果写gather_facts来控制是否收集远程系统的信息.如果不收集系统信息,那么上面的变量就不能在该playybook中使用了.

```
- hosts: whatever
  gather_facts: no
```

把运行结果当做变量使用-注册变量

把task的执行结果当作是一个变量的值也是可以的。这个时候就需要用到注册变量，将执行结果注册到一个变量中，待后面的action使用：

```
---

- hosts: web

  tasks:

     - shell: ls
       register: result
       ignore_errors: True

     - shell: echo "{{ result.stdout }}"
       when: result.rc == 5

     - debug: msg="{{ result.stdout }}"
```

文件模板中使用的变量

文件模板即template。Ansible使用的文件是python的一个j2的模板。

template变量的定义

在playbook中定义的变量，可以直接在template中使用。

下面的playbook脚本中使用了template module来拷贝文件index.html.j2，并且替换了index.html.j2中的变量为playbook中定义变量值。

```
---
- hosts: web
  vars:
    http_port: 80
    defined_name: "Hello My name is Jingjng"
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest

  - name: Write the configuration file
    template: src=templates/httpd.conf.j2 dest=/etc/httpd/conf/h
ttpd.conf
    notify:
    - restart apache

  - name: Write the default index.html file
    template: src=templates/index2.html.j2 dest=/var/www/html/in
dex.html

  - name: ensure apache is running
    service: name=httpd state=started
  - name: insert firewalld rule for httpd
    firewalld: port={{ http_port }}/tcp permanent=true state=ena
bled immediate=yes

  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

template变量的使用

在template index.html.j2中可以直接使用系统变量和用户自定义的变量

- 系统变量 **{{ ansible_hostname }}** , **{{ ansible_default_ipv4.address }}**
- 用户自定义的变量 **{{ defined_name }}**

index.html.j2文件：

```
<html>
<title>#46 Demo</title>

<!--
http://stackoverflow.com/questions/22223270/vertically-and-horiz
ontally-center-a-div-with-css
http://css-tricks.com/centering-in-the-unknown/
http://jsfiddle.net/6PaXB/
-->

<style>.block {text-align: center;margin-bottom:10px;}.block:bef
ore {content: '';display: inline-block;height: 100%;vertical-ali
gn: middle;margin-right: -0.25em;}.centered {display: inline-blo
ck;vertical-align: middle;width: 300px;}</style>

<body>
<div class="block" style="height: 99%;">
    <div class="centered">
        <h1>#46 Demo {{ defined_name }}</h1>
        <p>Served by {{ ansible_hostname }} ({{ ansible_default_
ipv4.address }}).</p>
    </div>
</div>
</body>
</html>
```

用命令行传递参数

定义命令行变量

在release.yml文件里，hosts和user都定义为变量，需要从命令行传递变量值。

```
---

- hosts: '{{ hosts }}'
  remote_user: '{{ user }}'

  tasks:
     - ...
```

使用命令行变量

在命令行里面传值得的方法：

```
ansible-playbook e33_var_in_command.yml --extra-vars "hosts=web
user=root"
```

还可以用json格式传递参数：

```
ansible-playbook e33_var_in_command.yml --extra-vars "{'hosts':'
vm-rhel7-1', 'user':'root'}"
```

还可以将参数放在文件里面：

```
ansible-playbook e33_var_in_command.yml --extra-vars "@vars.json
"
```

playbook也有逻辑控制的语句

- when： 条件判断语句，类似于变成语言中的if
- loop： 循环语句，类似于编程语言的中的while
- block： 把几个tasks组成一块代码，便于针对一组操作的异常处理等操作。

条件语句When

类似于编程语言的if

When语句

有时候用户有可能需满足特定条件才执行某一个特定的步骤。在某一个特定版本的系统上装包，或者只在磁盘空间满了的文件系统上执行清理操作一样。这些操作在Ansible上，使用 `when` 语句都非常简单.

主机为Debian Linux立刻关机

```
tasks:
  - name: "shutdown Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
```

根据action的执行结果，来决定接下来执行的action。

```
tasks:
  - command: /bin/false
    register: result
    ignore_errors: True
  - command: /bin/something
    when: result|failed
  - command: /bin/something_else
    when: result|success
  - command: /bin/still/something_else
    when: result|skipped
```

远程中的系统变量facts变量作为when的条件，用“|int”还可以转换返回值的类型：

```
---
- hosts: web
  tasks:
    - debug: msg="only on Red Hat 7, derivatives, and later"
      when: ansible_os_family == "RedHat" and ansible_lsb.major_
release|int >= 6
```

条件表达式

```
vars:
  epic: true
```

基本款

```
tasks:
    - shell: echo "This certainly is epic!"
      when: epic
```

否定款：

```
tasks:
    - shell: echo "This certainly isn't epic!"
      when: not epic
```

变量定义款

```
tasks:
    - shell: echo "I've got '{{ foo }}' and am not afraid to use
 it!"
      when: foo is defined

    - fail: msg="Bailing out. this play requires 'bar'"
      when: bar is not defined
```

数值表达款

```
tasks:
    - command: echo {{ item }}
      with_items: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

与Include一起用

```
- include: tasks/sometasks.yml
  when: "'reticulating splines' in output"
```

与Role一起用

```
- hosts: webservers
  roles:
     - { role: debian_stock_config, when: ansible_os_family == '
Debian' }
```

Loop循环

标准循环

为了保持简洁,重复的任务可以用以下简写的方式:

```
- name: add several users
  user: name={{ item }} state=present groups=wheel
  with_items:
     - testuser1
     - testuser2
```

如果你在变量文件中或者 ‘vars’ 区域定义了一组YAML列表,你也可以这样做:

```
vars:
  somelist: ["testuser1", "testuser2"]
tasks:
  -name: add several user
   user: name={{ item }} state=present groups=wheel
   with_items: "{{somelist}}"
```

使用 ‘with_items’ 用于迭代的条目类型不仅仅支持简单的字符串列表.如果你有一个哈希列表,那么你可以用以下方式来引用子项:

```
- name: add several users
  user: name={{ item.name }} state=present groups={{ item.groups
 }}
  with_items:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }
```

注意：如果同时使用 when 和 with_items （或其它循环声明），`when` 声明会为每个条目单独执行.请参见 the_when_statement 示例.

嵌套循环

循环也可以嵌套:

```
- name: give users access to multiple databases
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:ALL append
_privs=yes password=foo
  with_nested:
    - [ 'alice', 'bob' ]
    - [ 'clientdb', 'employeedb', 'providerd']
```

或者

```
- name: give users access to multiple databases
  mysql_user: name={{ item.0 }} priv={{ item.1 }}.*:ALL append_p
rivs=yes password=foo
  with_nested:
    - [ 'alice', 'bob' ]
    - [ 'clientdb', 'employeedb', 'providerd']
```

对哈希表使用循环

```
---
vars:
  alice:
    name: Alice Appleworth
    telephone: 123-456-7890
  bob:
    name: Bob Bananarama
    telephone: 987-654-3210
tasks:
  - name: Print phone records
    debug: msg="User {{ item.key }} is {{ item.value.name }} ({{
 item.value.telephone }})"
    with_dict: "{{users}}"
```

对文件列表使用循环

with_fileglob 可以以非递归的方式来模式匹配单个目录中的文件.如下面所示:

```
tasks:

    # first ensure our target directory exists
    - file: dest=/etc/fooapp state=directory

    # copy each file over that matches the given pattern
    - copy: src={{ item }} dest=/etc/fooapp/ owner=root mode=600
      with_fileglob:
        - /playbooks/files/fooapp/*
```

Block块

多个action组装成块，可以根据不同条件执行一段语句 ：

```
tasks:
     - block:
         - yum: name={{ item }} state=installed
           with_items:
             - httpd
             - memcached

         - template: src=templates/src.j2 dest=/etc/foo.conf

         - service: name=bar state=started enabled=True

       when: ansible_distribution == 'CentOS'
       become: true
       become_user: root
```

组装成块处理异常更方便：

```
tasks:
  - block:
      - debug: msg='i execute normally'
      - command: /bin/false
      - debug: msg='i never execute, cause ERROR!'
    rescue:
      - debug: msg='I caught an error'
      - command: /bin/false
      - debug: msg='I also never execute :-('
    always:
      - debug: msg="this always executes"
```

重用Playbook

不能站在巨人的肩膀上的编程语言不是好语言，支持重用机制会节省调研重复的工作上浪费大量的时间，当然也会提高可维护性。

Playbook的支持两种重用机制，一种是简单的直接重新利用静态的Playbook脚本，例外一种是类似于编程语言中函数的机制。

- include语句 - 重用静态的Playbook脚本，使用起来简单、直接。
- role语言 - Playbook的“函数机制”，使用方法稍复杂、功能强大。是Playbook脚本的共享平台ansible galaxy主要的分享方式。

Include语句

Include语句的功能，基本的代码重用机制。主要重用tasks。同时Include可将tasks分割成多个文件，避免Playbook过于臃肿，使用户更关注于整体的架构，而不是实现的细节上。

普通用法

像其它语言的Include语句一样，直接Include：

```
---
# possibly saved as tasks/firewall_httpd_default.yml

  - name: insert firewalld rule for httpd
    firewalld: port=80/tcp permanent=true state=enabled immediat
e=yes
```

main.yml文件中调用include的方法:

```
tasks:
    - include: tasks/firewall_httpd_default.yml
```

高级用法-使用参数

include文件中还可以定义参数

被include的文件tasks/firewall_httpd_default.yml中，使用 `{{ port }}` 定义了一个名字为port的参数。

```
---
  - name: insert firewalld rule for httpd
    firewalld: port={{ port }}/tcp permanent=true state=enabled
immediate=yes
```

传参数的各种方法

- 在执行的playbook传参数，可以加在行尾，使用空格分隔：

```
tasks:
  - include: tasks/firewall.yml port=80
  - include: tasks/firewall.yml port=3260
  - include: tasks/firewall.yml port=423
```

- 还可以使用yml的字典传参数：

```
tasks:

  - include: wordpress.yml
    vars:
        wp_user: timmy
        ssh_keys:
          - keys/one.txt
          - keys/two.txt
```

- 还可以把一条task简写成成一个类JSON的形式传参数：

```
tasks:
 - { include: wordpress.yml, wp_user: timmy, ssh_keys: [ 'keys/one.txt', 'keys/two.txt' ] }
```

- 当然在playbook中已经定义了的参数，就不需要再显示传入值了，可以直接写成下面的：

```
---
- hosts: lb
  vars:
    port: 3206
  remote_user: root
  tasks:
    - include: tasks/firewall.yml
```

include语句相关的那些坑

- 在handlers里面加include

  handlers中加入include语句的语法如下：

  ```
  handlers:
    - include: handlers/handlers.yml
  ```

  然而为什么有一处文档里面写可以调用。文档下面两个地方提到include里面的handlers，但是两处是矛盾的:

  - hander的文档写不能调用 http://docs.ansible.com/ansible/playbooks_intro.html
  - include的文档写能调用 http://docs.ansible.com/ansible/playbooks_roles.html#task-include-files-and-encouraging-reuse

  通过下面的例子实测后，基于ansible1.9是不能调用include里面的handler的，不过基于ansible2.0+是可以调用include里面的handler的。所以在使用的时候注意你安装的ansible版本。

  ```

  - hosts: lb user: root gather_facts: no vars:

    ```
    random_number: ""
    ```

    tasks:

    - name: Copy the /etc/hosts to /tmp/hosts. copy: src=/etc/hosts dest=/tmp/hosts. notify:
      - restart apache
      - restart apache in handlers
    ```

```
handlers:
  - include: handlers/handlers.yml
  - name: restart apache
    debug: msg="This is the handler restart apache"
```

```
* Ansible允许的全局（或者叫plays）加include

  然而这种使用方式并不推荐，首先它不支持嵌入include，而且很多playbook的参数也不可以使用。
```

- name: this is a play at the top level of a file hosts: all remote_user: root

  tasks:

  - name: say hi tags: foo shell: echo "hi..."

全局include，或者叫playbook include

- include: load_balancers.yml
- include: webservers.yml
- include: dbservers.yml ```

- 为了使include功能更强大，在每个新出的ansible都会添加新的一些功能，例如在2.0中添加了include动态名字的yml，然而这样的用法有很多的限制，不够成熟，可能在更新的ansible又去掉了，学习和维护成本很高。所以需要使用更灵活的重用机制时，建议用下一节介绍的role。

Role - Playbook的“Package”

Role比include更强大灵活的代码重用和分享机制。Include类似于编程语言中的include，是重用单个文件的，重用的功能有限。

而Role类似于编程语言中的“Package”，可以重用一组文件形成完整的功能。例如安装和配置apache，需要tasks实现安装包和拷贝模版等，httpd.conf和index.html的模版文件，和handler文件实现重起功能。这些文件都可以放在一个role里面，供不同的playbook文件重用。

Ansible非常提倡在playbook中使用role，并且提供了一个分享role的平台Ansible Galaxy, https://galaxy.ansible.com/, 在galaxy上可以找到别人写好的role。在后面的章节中，我们再详细介绍如果使用它。

定义role完整的目录结构

在**ansible**中,通过遵循特定的目录结构,就可以实现对**role**的定义。。具体遵循的目录结构是什么呢？看下面的例子：

下面的目录结构定义了一个role：名字为myrole。在site.yml，调用了这个role。

<table>
<tr><th>role的目录结构</th><th>site.yml中调用role</th></tr>
<tr><td>
<pre>
site.yml
roles/
├── myrole
    ├── tasks
    │   └── main.yml
    ├── handlers
    │   └── main.yml
    ├── defaults
    │   └── main.yml
    ├── vars
    │   └── main.yml
    ├── files
    ├── templates
    ├── README.md
    ├── meta
    │   └── main.yml
    └── tests
        ├── inventory
        └── test.yml
</pre>
</td><td>
<pre>

- hosts: webservers
  roles:
     - myrole
</pre>
</td></tr>
</table>

ansible并不要求role包含上述所有的目录及文件，根据role的功能需要加入对应的目录和文件。下面是每个目录和文件的功能。

- 如果 roles/x/tasks/main.yml 存在, 其中列出的 tasks 将被添加到 play 中，所以这个文件也可以视作role的入口文件，想看role做了什么操作，可以从此文件看起。
- 如果 roles/x/handlers/main.yml 存在, 其中列出的 handlers 将被添加到 play 中
- 如果 roles/x/vars/main.yml 存在, 其中列出的 variables 将被添加到 play 中
- 如果 roles/x/meta/main.yml 存在, 其中列出的 “角色依赖” 将被添加到 roles 列表中
- roles/x/tasks/main.yml中所有tasks，可以引用 roles/x/{files,templates,tasks}中的文件，不需要指明文件的路径。

你自己在写role的时候，一般都要包含role入口文件roles/x/tasks/main.yml，其它的文件和目录，可以根据需求选择加入。

学会写一个完整功能的role是一个想对复杂的过程，也是ansible中稍高级的使用方法。在本小节，我们重点介绍如何使用别人已经写好的role。在后面的章节中，我们会通过具体的示例来逐步介绍写role的所需的知识。

带参数的Role

参数在role中是如何定义的呢

定义一个带参数的role,名字是myrole,那么目录结构为

```
main.yml
roles
  role_with_var
    tasks
      main.yml
```

在roles/myrole/tasks/main.yml中,使用 `{{ }}` 定义的变量就可以了

```
---
- name: use param
  debug: msg="{{ param }}"
```

使用带参数的role

那么在main.yml就可以用如下的方法使用myrole

```
---

- hosts: webservers
  roles:
    - { role: myrole, param: 'Call some_role for the 1st time' }
    - { role: myrole, param: 'Call some_role for the 2nd time' }
```

或者写成yml字典格式：

```
---

- hosts: webservers
  roles:
    - role: myrole
      param: 'Call some_role for the 1st time'
    - role: myrole
      param: 'Call some_role for the 2nd time'
```

role指定默认的参数

指定默认参数后,如果在调用时传参数了,那么就使用传入的参数值.如果调用的时候没有传参数,那么就使用默认的参数值.

指定默认参数很简单,以上面的role_with_var为例

```
main.yml
roles:
  myrole
    tasks
      main.yml
    defaults
      main.yml
```

在roles/myrole/defaults/main.yml中,使用yml的字典定义语法定义param的值,如下:

```
param: "I am the default value"
```

这样在main.yml中,下面两种调用方法都可以

```
---

- hosts: webservers
  roles:
    - role_with_var
    - { role: role_with_var, param: 'I am the value from externa
l' }
```

更多的例子在https://github.com/shijingjing1221/ansible-first-book-examples/blob/master/role_vars.yml 中

role与条件语句when一起执行

下面的例子中,my_role只有在RedHat系列的server上才执行。

```
---

- hosts: webservers
  roles:
    - { role: my_role, when: "ansible_os_family == 'RedHat'" }
```

同样也可以写成yml字典格式

```
---

- hosts: webservers
  roles:
    - role: my_role
      when: "ansible_os_family == 'RedHat'"
```

roles和tasks的执行顺序

如果一个playbook同时出现role和tasks，他们的调用顺序是什么样的呢？

先揭晓答案，在根据实例来验证：

pre_tasks > role > tasks > post_tasks

```
---

- hosts: lb
  user: root

  pre_tasks:
    - name: pre
      shell: echo 'hello'

  roles:
    - { role: some_role }

  tasks:
    - name: task
      shell: echo 'still busy'

  post_tasks:
    - name: post
      shell: echo 'goodbye'
```

执行的结果为：

```
PLAY [lb] ******************************************************
****************


TASK [setup] ***************************************************
****************
ok: [rhel7u3]

TASK [pre] *****************************************************
****************
changed: [rhel7u3]

TASK [some_role : some role] ***********************************
****************
ok: [rhel7u3] => {
    "msg": "Im some role"
}

TASK [task] ****************************************************
****************
changed: [rhel7u3]

TASK [post] ****************************************************
****************
changed: [rhel7u3]

PLAY RECAP *****************************************************
****************
rhel7u3                    : ok=5    changed=3    unreachable=0
   failed=0
```

用tags实现执行playbook中部分tasks

如果playbook文件比较大，在执行的时候只是想执行部分功能，这个时候没有有解决方案呢？Playbook提供了tags变迁可以实现部分运行。

tags的基本用法

例如，文件example.yml如何所示，标记了两个tag：packages和configuration

```
tasks:

  - yum: name={{ item }} state=installed
    with_items:
       - httpd
    tags:
       - packages

  - name: copy httpd.conf
    template: src=templates/httpd.conf.j2 dest=/etc/httpd/conf/h
ttpd.conf
    tags:
       - configuration

  - name: copy index.html
    template: src=templates/index.html.j2 dest=/var/www/html/ind
ex.html
    tags:
       - configuration
```

- 那么我们在执行的时候，如果不加任何tag参数，那么会执行所有的tasks

  ```
  ansible-playbook example.yml
  ```

- 指定执行安装部分的tasks，则可以利用关键字tags

```
ansible-playbook example.yml --tags "packages"
```

- 指定不执行packages部分的task，则可以利用关键字skip-tags

```
ansible-playbook example.yml --skip-tags "configuration"
```

特殊的Tags

- “always”

  tags的名字是用户自定义的，但是如果你把tags的名字定义为“always”，那么就有点特别了。只要在执行playbook时，没有明确指定不执行always tag，那么它就会被执行。

  在下面的例子中，即使你只指定执行packages，那么always也会被执行。

```
tasks:

  - debug: msg="Always print this debug message"
    tags:
      - always

  - yum: name= state=installed
    with_items:
       - httpd
    tags:
       - packages

  - template: src=templates/httpd.conf.j2 dest=/etc/httpd/co
nf/httpd.conf
    tags:
       - configuration
```

  指定运行packages时，还是会执行always tag对应的tasks

```
ansible-playbook tags_always.yml --tags "packages"
```

- “tagged”，“untagged”和“all”

```
tasks:

  - debug: msg="I am not tagged"
    tags:
      - tag1

  - debug: msg="I am not tagged"
```

分别指定--tags为“tagged”，“untagged”和“all”试下效果吧：

```
ansible-playbook tags_tagged_untagged_all.yml --tags tagged
```

```
ansible-playbook tags_tagged_untagged_all.yml --tags untagge
d
```

```
ansible-playbook tags_tagged_untagged_all.yml --tags all
```

在include中和role中使用tags

include语句指定执行的tags的语法：

```
- include: foo.yml
  tags: [web,foo]
```

调用role中的tags的语法为：

```
roles:
  - { role: webserver, port: 5000, tags: [ 'web', 'foo' ] }
```

更多的Ansible模块

- 介绍两类Modules： Core Module和Extra module
- Extra module的配置和使用方法
- 通过命令行查看modules的用法

Modules的分类

你Ansible Module文档上查看单个Module的时候,每一个Module文档的底部都会标识, 这是一个"Core Module", 或者这是一个"Extra Module".

比如, yum就是一个core module, yum_repository就是一个extra module,

Core Module

- 不需要格外下载和配置，安装ansible后就可以直接使用的.
- 比较常用的module
- 经过严格测试的.

Extra module

- 进行下载和格外的配置才能使用
- 次常用的
- 还有可能存在bug的

Extra module的使用方法

使用Exra module需要进行下面的配置，就可以在命令行或者是playbook中使用了。配置后extra module使用方法和core module的使用方法是一样的。

[注]Ansible 2.3以后，Extra module的使用就和core module一样了，无需任何额外的配置，直接在playbook和命令行中使用。其实Ansible团队会一直致力于把成熟的长期使用没有问题的Module放入Core Module中，方便客户的使用。所以当你的Playbok运行报错是没有相应的module时，你只要心中有数可能出现问题的地方和解决方案就可以。

1 下载**ansible module extra**项目

```
git clone https://github.com/ansible/ansible-modules-extras.git
```

我的一下在/home/jshi/software/目录下了，后面会用到这个目录。

2 修改配置文件或者环境变量

方法**1 -** 改**ansible**默认配置文件**/etc/ansible/ansible.cfg**

修改ansible配置文件/etc/ansible/ansible.cfg, 添加一行

```
library    = /home/jshi/software/ansible-modules-extras/
```

方法**2 -** 改**ansible**当前目录下配置文件**ansible.cfg**

改ansible playbook当前的目录下的配置文件ansible.cfg，那么只对当前目录的**playbook**生效。对所有其它目录，包括父目录和子目录的playbook都不生效。

```
library/ansible-modules-extras
ansible.cfg
use_extra_module.yml
subfolder/use_extra_module_will_throw_error.yml
```

在当前目录的ansible.cfg中，可以使用相对路径：

```
library = library/ansible-modules-extras/
```

方法**3 -** 该环境变量

```
export ANSIBLE_LIBRARY=/project/demo/demoansible/library/ansible
-module-extras
```

如果需要在重启后生效，那么放在~/.bashrc中声明ANSIBLE_LIBRARY变量：

```
$ echo >>~/.bashrc <<EOF

export ANSIBLE_LIBRARY=/project/demo/demoansible/library/ansible
-module-extras

EOF

$ source ~/.bashrc
```

命令行查看module的用法

类似bash命令的man，ansible也可以通过命令行查看module的用法。命令是ansible-doc，语法如下:

```
ansible-doc module_name
```

core module可以在任何目录下执行。例如查看yum的用法：

```
ansible-doc yum
```

extra module必须在配置了extra module的目录下查看用法(行为当前目录下的playbook是一致的):

```
ansible-doc yum_repository
```

最佳使用方法

- 鼓励文件的重用，尽量使用include和role避免重复的代码。
- 尽量把大的文件分成小的文件

https://github.com/ansible/ansible-examples

```
production                # inventory file for production server
s
staging                   # inventory file for staging environme
nt

group_vars/
   group1                 # here we assign variables to particul
ar groups
   group2                 # ""
host_vars/
   hostname1              # if systems need specific variables,
put them here
   hostname2              # ""

library/                  # if any custom modules, put them here
 (optional)
filter_plugins/           # if any custom filter plugins, put th
em here (optional)

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier

roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller
files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
```

```
        templates/        #  <-- files for use with the template
 resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy res
ource
            foo.sh        #  <-- script files for use with the s
cript resource
        vars/             #
            main.yml      #  <-- variables associated with this
role
        defaults/         #
            main.yml      #  <-- default lower priority variable
s for this role
        meta/             #
            main.yml      #  <-- role dependencies

    webtier/              # same kind of structure as "common" w
as above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

推荐的参考资料

Ansible英文视频教程: https://sysadmincasts.com/episodes/43-19-minutes-with-ansible-part-1-4

本书中所有playbook的例子都在github上：
https://github.com/shijingjing1221/ansible-first-book-examples/

YAML语法基础

文件开始符

```
---
```

数组List

```
- element1
- element2
- element3
```

数组中的每个元素都是以 - 开始的。

字典(Hash or Directory)

```
key: value
```

key和value已冒号加空格分隔。

复杂的字典

字典的嵌套

```
# An employee record
martin:
    name: Martin D'vloper
    job: Developer
    skill: Elite
```

字典和数组的嵌套

```
-  martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
-  tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

注意的地方

变量里有：要加引号

```
foo: "somebody said I should put a colon here: so I did"
```

变量的引用要加引号

```
foo: "{{ variable }}"
```

参考资料

https://en.wikipedia.org/wiki/YAML

http://www.yamllint.com/

待续

Ansible Tower 简介

https://www.ansible.com/tower

编写自己的Ansible module